Measuring processor performance

- Pipelines – improve performance
- Programs – must be optimized
- Caches – improve performance
- Benchmarks – measure performance
- Device Fabrication – improves performance
- Various optimizations are possible

Performance (out-of-order processing)

Pipelining provides higher performance by allowing execution of different instructions to overlap.

Pipeline performance is improved by allowing instructions to execute out of order, (**OOO**) improving the average number of completed *instructions per cycle*(**IPC**).

A superscalar processor can achieve an IPC of greater than 1 by allowing multiple instructions to go through the pipeline in parallel. Superscalar designs are described by their issue width, the maximum number of instructions that can enter the pipeline simultaneously.

A longer pipeline allows more instructions in the pipe at one time. Doubling frequency requires increasing pipeline depth by more than a factor of 2. Even if frequency is doubled in this fashion, performance will not double because IPC drops as pipeline depth increases.

$$\text{Frequency} = \frac{1}{T_{\text{cycle}}} = \frac{1}{\frac{T_{\text{logic}}}{\text{depth}} + T_{\text{overhead}}} = \frac{\text{depth}}{T_{\text{logic}} + \text{depth} \times T_{\text{overhead}}}$$

Performance

Better performance simply means less program run time.

$$\text{Performance} \propto \frac{1}{\text{run time}} = \frac{\text{frequency} \times \text{instructions per cycle}}{\text{instruction count}}$$

To improve performance, we must increase frequency or average instructions per cycle (IPC) or reduce the number of instructions required.

For the ideal case of no delay overhead and no added stalls, doubling pipeline depth will double performance. The real improvement depends upon how much circuit design minimizes the delay overhead per pipestage and how much microarchitectural improvements offset the reduction in IPC.

$$\text{Performance} \propto \text{frequency} \times \text{IPC} =$$

$$\frac{\text{depth}}{(T_{\text{logic}} + \text{depth} \times T_{\text{overhead}})(1 + \text{stalls per instruction})}$$

The processor cycle time will be set by the slowest pipestage. To prevent instructions requiring more computation from limiting processor frequency, they are designed to execute over more pipestages.

Performance

The most important rule in designing for high performance is to **make the common case fast**. This idea is sometimes formally referred to as **Amdahl's law**, which estimates the overall speedup obtained by decreasing the time spent on some fraction of a job.

Amdahl's law:

$$\text{Overall speedup} = \frac{\text{old time}}{\text{new time}} = \frac{1}{1 - \text{fraction}} + \frac{\text{fraction}}{\text{speedup}}$$

Frequency is the same for all instructions, so Amdahl's law applies only to improvements in IPC. More functional units allow higher issue width. Better reordering algorithms reduce pipeline stalls. More specialized logic reduces the latency of computation. Larger caches can reduce the average latency of memory accesses. All of these microarchitectural changes improve IPC but at the cost of design complexity and die area.

Performance

Fred Pollack compared the relative performance of different Intel processors and found that in the same fabrication technology, performance improved roughly with the square root of added die area. This relationship is often called *Pollack's rule.*[2]

Pollack's rule:

$$\text{Performance} \propto \sqrt{\text{die area}}$$

This rule of thumb means that a new microarchitecture that requires twice the die area will likely provide only a 40 percent improvement in performance.

Performance

Because values like MIPS, MFLOPS, and IPC will all vary from one program to another, one of the most commonly used measures of performance is the processor frequency.

Benchmarks (SPEC) are the best way of measuring processor performance, but they are far from perfect. The biggest problems are choosing benchmark programs, compiler optimizations, and system configurations. Different choices for any of these may suddenly change which computer scores as the fastest on a given benchmark.

Processors with large caches tend to look better running older benchmarks.

Processors with complex hardware reordering look relatively better when making comparisons with simple compilers.

Ultimately any benchmark measures not only processor performance but also computer performance. It is impossible for the processor to run any program at all without the support of the chipset and main memory, but these components are also important in the overall computer performance.

Performance & Caches

- Caches
  - Enable design for common case: cache hit
    - Cycle time, pipeline organization
    - Recovery policy
  - Uncommon case: cache miss
    - Fetch from next level
      - Apply recursively if multiple levels
    - What to do in the meantime?
- Q1 - What is performance impact?
- Various optimizations are possible

Performance Impact

- Cache hit latency figures(CPI)
  - Included in “pipeline” portion of CPI
    - E.g. IBM study: 1.15 CPI with 100% cache hits
  - Typically 1-3 cycles for L1 cache
    - Intel/HP McKinley: 1 cycle, latency factors:
      - Heroic array design
      - No address generation: load r1, (r2)
    - IBM Power4: 3 cycles, latency factors:
      - Address generation
      - Array access
      - Word select and align

CPU Performance

- Basic problem: how can we tell if one processor is faster than another?

- Execution time is analogous to performance

$$\frac{Performance_X}{Performance_Y} = \frac{Execution\ Time_Y}{Execution\ Time_X} = n$$

- Execution time (per processor core) =

$$\begin{matrix} CPU\ execution\ time \\ for\ a\ program \end{matrix} = \begin{matrix} CPU\ clock\ cycles \\ for\ a\ program \end{matrix} \times Clock\ cycle\ time$$

$$\begin{matrix} CPU\ execution\ time \\ for\ a\ program \end{matrix} = \frac{CPU\ clock\ cycles\ for\ a\ program}{Clock\ rate}$$

Performance Factors

- *Execution time =*

  *instructions per program * cycles per instruction * seconds per cycle*

- Problems:
  - Cycles per instruction depends on many factors
    - i.e. instruction mix, I/O, memory

- Performance factors:
  - Algorithm ➔ instruction count, CPI
  - Programming language ➔ instruction count, CPI
  - Compiler ➔ instruction count, CPI
  - ISA ➔ instruction count, clock rate, CPI

Performance Measures

- CPI = cycles per instruction
  - CPI provides one way of comparing two different implementations of the same instruction set architecture
  - CPI is tricky!
    - Instructions from the same ISA require different number of cycles
    - Depends on program
    - Memory behavior affects CPI
  - CPI is *usually* defined for a particular program on a particular CPU

$$CPU\ time = Instruction\ count \times CPI \times Clock\ cycle\ time$$

$$CPU\ time = \frac{Instruction\ count \times CPI}{Clock\ rate}$$

$$CPU\ clock\ cycles = Instructions\ for\ a\ program \times Average\ CPI$$

Example

- Suppose one machine, A, executes a program with an average CPI of 2.1.
- Suppose another machine, B (with the same instruction set and an enhanced compiler), executes the same program with 25% less instructions and with a CPI of 1.8 at 800MHz.
- In order for the two machines to have the same performance, what does the clock rate of the first machine (machine A) need to be?

Example

- instructionsA * CPIA * (1/clockrateA) = instructionsB * CPIB * (1/clockrateB)
- instructionsA * 2.1 * (1/clockrateA) = .75*instructionsA * 1.8 * (1/800x10^6)
- 2.1 * (1/clockrateA) = .75*1.8 * (1/800x10^6) {*solve for clockrateA*}
- clockrateA = 1.2444GHz

Performance Measures

- Program execution time vs CPU execution time
  - Must account for other programs and I/O
  - User CPU time
  - CPU time spent executing a program

- System CPU time
  - Amount of CPU time the operating system spends on behalf of a program
  - Ex. system calls, such as I/O, synchronization, etc.

Example

Suppose a program has the following instruction classes, CPIs, and mixtures:

Instruction type	CPI	ratio
A	1.4	55%
B	2.4	15%
C	2.0	30%

Your CPU design engineers give you the following options:
Option A: Reduce the CPI of instruction type A to 1.1

(.55)*1.1 + (.15)*2.4 + (.30)*2 = 1.565

Option B: Reduce the CPI of instruction type B to 1.2

(.55)*1.4 + (.15)*1.2 + (.30)*2 = 1.55

Which option would you choose and why?

B, yields lower overall CPI